Deep Learning Mathematics Atlas

深度学习数学图鉴

从 *PyTorch* 算子到 *LaTeX* 定义的硬核映射

Antigravity & Sisyphus & 吴东泽

February 21, 2026

Edition 2.0

Contents

前言 (Preface)

深度学习在工程上的辉煌胜利，往往掩盖了其底层严谨的数学逻辑。对于当今的 AI 开发者/刚入学的研究生/想做深度学习项目的人/对这方面感兴趣的本科生而言，我们正面临着一个普遍的困境：**黑盒化的陷阱**。

当我们敲下 `nn.Linear` 或 `nn.CrossEntropyLoss` 时，PyTorch 等现代框架以极其优雅的 API 替我们屏蔽了复杂的矩阵乘法、张量偏导与梯度流向。这种工程上的便利极大降低了入门门槛，但也让许多从业者逐渐沦为“调包侠”。

然而，当你试图复现一篇顶会论文（如 LLaMA 的旋转位置编码 RoPE，或是 Mamba 的选择性状态空间），当你遭遇训练过程中诡异的 NaN（梯度爆炸），或是当你想手写 CUDA 算子进行极限推理加速时，你会痛苦地发现：**仅仅读懂 Python 代码已经远远不够了**。

学术界的论文（Paper）是用纯粹的 LaTeX 数学语言写成的，而工程界的落地则是用 Python 和 C++ 堆砌的。这两者之间存在着巨大的认知鸿沟。本书正是为了打破这堵高墙，打造一块连接学术界与工程界的**罗塞塔石碑 (Rosetta Stone)**。

在此版本中，我们引入了全新的“左右对照”排版风格，旨在提供更直观的数学与代码映射体验。

Antigravity & Sisyphus & 吴东泽
2026 年 2 月

Part I

基石篇 (Foundations)

Chapter 1

张量基础运算

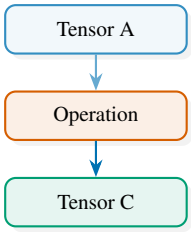


Figure 1.1: Tensor Operation Flow

在深度学习中，张量（Tensor）是承载数据的基本容器。所有的复杂层最终都会分解为对这些多维数组的基础算子。

1.1 逐元素运算 (Element-wise Operations)

1.1.1 Hadamard 乘积

逐元素乘法是指两个形状相同的张量，对应位置的元素相乘。

$$C = A \odot B \quad \text{其中} C_{i,j} = A_{i,j} \times B_{i,j} \tag{1.1}$$

`torch.mul(A, B)` 或 `A * B`

1.1.2 幂运算

对张量中的每一个分量求 n 次幂。

$$y = x^n \tag{1.2}$$

`torch.pow(x, n)`

1.2 线性代数 (Linear Algebra)

1.2.1 矩阵乘法

这是神经网络中线性变换的核心，常用于全连接层和注意力机制。

$$C = AB \implies C_{i,j} = \sum_{k=1}^{n} A_{i,k} B_{k,j} \quad (1.3)$$

`torch.matmul(A, B)` 或 `A @ B`

1.2.2 转置

交换张量的维度，在矩阵中通常指行与列的互换。

$$A^T_{i,j} = A_{j,i} \quad (1.4)$$

`tensor.T` 或 `tensor.t()`

1.3 统计规约 (Reduction)

1.3.1 L2 范数

计算向量或矩阵的模长。

L2

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2} \quad (1.5)$$

`torch.norm(x, p=2)`

Chapter 2

高级线性代数算子

线性代数算子是深度学习底层的动力引擎。除了基础的矩阵乘法外，诸如奇异值分解（SVD）、特征值分解和爱因斯坦求和约定（Einsum）在现代模型压缩、稳定训练和高性能计算中扮演着至关重要的角色。

2.1 矩阵分解 (Matrix Decomposition)

2.1.1 奇异值分解 (SVD)

SVD 将任何矩阵分解为旋转、缩放和旋转三个过程。在深度学习中，常用于 LoRA 的权重初始化或模型压缩。

(SVD)

$$A = U\Sigma V^T \tag{2.1}$$

其中：

- U, V 是正交矩阵。
- Σ 是对角矩阵，包含奇异值。

```
U, S, Vh = torch.linalg.svd(A)
```

2.1.2 QR 分解

QR 分解将矩阵分解为一个正交矩阵 Q 和一个上三角矩阵 R。常用于数值稳定性要求极高的线性方程组求解。

QR

$$A = QR \tag{2.2}$$

其中 $Q^TQ = I$，R 为上三角。

```
Q, R = torch.linalg.qr(A)
```

2.2 矩阵属性与度量

2.2.1 行列式与逆

虽然在反向传播中较少直接求逆，但在概率图模型或雅可比行列式计算中不可或缺。

$$\det(A) \cdot A^{-1} = \text{adj}(A) \tag{2.3}$$

```
torch.linalg.det(A)
torch.linalg.inv(A)
```

2.2.2 矩阵范数 (Matrix Norm)

用于度量矩阵的“大小”，在谱归一化（Spectral Normalization）中特指算子范数。

(Spectral Norm)

$$\|A\|_2 = \sigma_{\max}(A) \tag{2.4}$$

即矩阵最大奇异值。

```
torch.linalg.matrix_norm(A,
ord=2)
```

2.3 高级张量约定

2.3.1 爱因斯坦求和约定 (Einsum)

Einsum 是处理多维张量收缩的“瑞士军刀”，能以极简语法实现转置、乘法、迹和外积。

Einstein Summation

$$C_{ij} = \sum_k A_{ik} B_{kj} \implies \texttt{ik,kj->ij} \tag{2.5}$$

```
torch.einsum('ik,kj->ij', A, B)
```

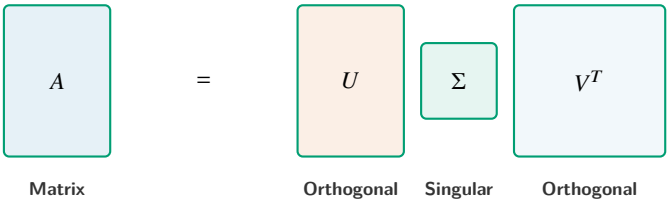
2.3.2 批量矩阵乘法 (BMM)

在处理 Batch 数据（如 Attention 权重）时，高效处理前两个维度外的矩阵乘法。

Batch MatMul

$$C_{b,i,j} = \sum_k A_{b,i,k} B_{b,k,j} \tag{2.6}$$

```
torch.bmm(A, B)
```



Chapter 3

概率论基础 (Probability Foundations)

概率论是深度学习处理不确定性、构建生成模型和设计损失函数的基石。所有的变分推断、扩散模型和信息论指标都建立在概率分布的基础之上。

3.1 常见概率分布 (Common Distributions)

3.1.1 正态分布 (Normal Distribution)

正态分布（高斯分布）是自然界中最常见的连续概率分布，也是许多生成模型（如 VAE 和 Diffusion）的先验分布假设。

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\top}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) \tag{3.1}$$

```
torch.distributions.MultivariateNormal(loc, covariance_matrix)
```

3.1.2 伯努利与多项式分布 (Bernoulli & Categorical)

离散分布常用于分类任务的标签建模。

(Categorical Distribution)

$$P(X=k) = p_k, \quad \sum_{k=1}^{K} p_k = 1 \tag{3.2}$$

```
torch.distributions.Categorical(probs=p)
```

3.2 贝叶斯定理与变分推断 (Bayesian & VI)

贝叶斯定理是生成模型（如 VAE）处理潜在变量（Latent Variables）的核心理论。

(Bayes' Theorem)

$$P(Z|X) = \frac{P(X|Z)P(Z)}{P(X)} \tag{3.3}$$

其中 $P(Z|X)$ 是后验，$P(X|Z)$ 是似然，$P(Z)$ 是先验。

在深度学习中，我们通常使用编码器 $q_\phi(Z|X)$ 来近似难以计算的真后验 $P(Z|X)$。

3.3 马尔可夫链 (Markov Chains)

马尔可夫链描述了一个状态空间中经过从一个状态到另一个状态的转换的随机过程。该过程要求具备“无记忆”的性质（即马尔可夫性质）。

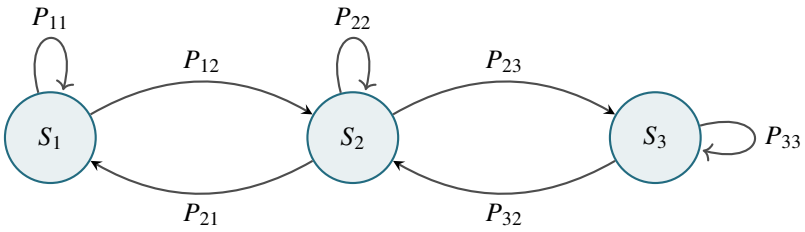


Figure 3.1: 简单的马尔可夫链状态转移图

马尔可夫性质的数学定义为：

$$P(X_{n+1} = x|X_n = x_n, X_{n-1} = x_{n-1}, \ldots, X_0 = x_0) = P(X_{n+1} = x|X_n = x_n) \tag{3.4}$$

Chapter 4

激活函数

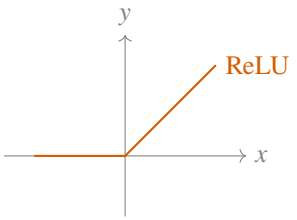


Figure 4.1: ReLU Function

激活函数通过引入非线性变换，使得神经网络能够拟合极其复杂的函数分布。

4.1 常用激活函数

4.1.1 ReLU (Rectified Linear Unit)

ReLU 是深度学习中最常用的激活函数，因其计算简单且能有效缓解梯度消失问题。

ReLU

$$\text{ReLU}(x) = \max(0, x) \tag{4.1}$$

`torch.relu(x)` 或 `nn.ReLU()`

4.1.2 Sigmoid

将输入映射到 $(0, 1)$ 区间，常用于二分类问题的概率预测。

Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.2}$$

`torch.sigmoid(x)` 或 `nn.Sigmoid()`

4.1.3 Tanh (双曲正切)

将输入映射到 $(-1, 1)$ 区间，输出均值为 0，在某些网络中比 Sigmoid 收敛更快。

Tanh

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4.3}$$

`torch.tanh(x)` 或 `nn.Tanh()`

4.1.4 Softmax

常用于多分类任务的输出层，将一组向量转化为概率分布。

Softmax

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}} \tag{4.4}$$

```
torch.softmax(x, dim)
```

Part II

解剖篇 (Anatomy)

Chapter 5

神经网络层

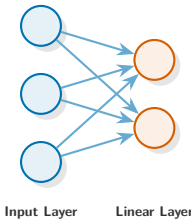


Figure 5.1: Linear Layer (Fully Connected)

神经网络层是参数的学习单元，它将基础算子封装成带有可学习权重的模块。

5.1 线性变换与卷积

5.1.1 线性层 (Linear / Fully Connected)

执行仿射变换，是多层感知机（MLP）的核心。

$$\mathbf{y} = \mathbf{x}\mathbf{W}^T + \mathbf{b} \tag{5.1}$$

其中 $\mathbf{W} \in \mathbb{R}^{out \times in}$ 是权重矩阵，$\mathbf{b} \in \mathbb{R}^{out}$ 是偏置向量。

```
nn.Linear(in_features,
    out_features)
```

5.1.2 2D 卷积层 (Convolutional Layer)

卷积层通过局部连接和权值共享提取空间特征。注意：PyTorch 实际执行的是互相关运算。

2D

输出尺寸计算公式:

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor \tag{5.2}$$

互相关运算定义:

$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out_j}, k) \star \text{input}(N_i, k) \tag{5.3}$$

其中 $\star$ 在此表示离散互相关：$(f \star g)(n) = \sum_m f(m) g(n+m)$。

```
nn.Conv2d(in_channels,
out_channels, kernel_size,
stride, padding)
```

5.1.3 池化层 (Pooling Layers)

用于下采样，减少计算量并增强平移不变性。

(Max Pooling)

$$\text{out}(C, h, w) = \max_{m,n} \text{input}(C, h \times S + m, w \times S + n) \tag{5.4}$$

其中 m, n 在卷积核大小范围内滑动。

```
nn.MaxPool2d(kernel_size)
```

Chapter 6

循环与序列层

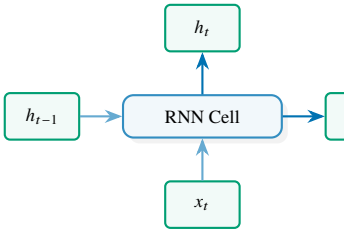


Figure 6.1: RNN Time Step

针对序列数据，神经网络需要具备“记忆”能力，通过隐藏状态在时间步之间传递信息。

6.1 RNN (Recurrent Neural Network)

最基础的循环单元，由于梯度消失问题，难以处理长序列。

RNN

$$h_t = \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{t-1} + b_{hh}) \tag{6.1}$$

```
nn.RNN(input_size, hidden_size)
```

6.2 LSTM (Long Short-Term Memory)

通过引入“细胞状态”（Cell State）和门控机制，有效解决了长距离依赖问题。

LSTM

更新过程:

```
nn.LSTM(input_size, hidden_size)
```

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \quad \text{(输入门)} \tag{6.2}$$

$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \quad \text{(遗忘门)} \tag{6.3}$$

$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \quad \text{(细胞候选)} \tag{6.4}$$

$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \quad \text{(输出门)} \tag{6.5}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad \text{(细胞状态更新)} \tag{6.6}$$

$$h_t = o_t \odot \tanh(c_t) \quad \text{(隐藏状态更新)} \tag{6.7}$$

6.3 GRU (Gated Recurrent Unit)

LSTM 的简化版本，合并了门控，计算效率更高。

GRU

```
nn.GRU(input_size, hidden_size)
```

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}) \quad \text{(重置门)} \tag{6.8}$$

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}) \quad \text{(更新门)} \tag{6.9}$$

$$n_t = \tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{t-1} + b_{hn})) \tag{6.10}$$

$$h_t = (1 - z_t) \odot n_t + z_t \odot h_{t-1} \tag{6.11}$$

Chapter 7

归一化层

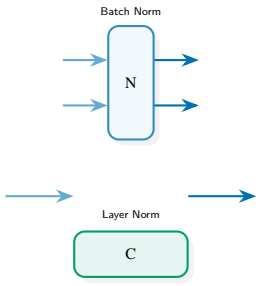


Figure 7.1: BN vs LN Dimension

归一化技术通过重新调整神经元输入的分布，加速模型收敛并提高训练稳定性。

7.1 Batch Normalization (批归一化)

针对整个 Mini-batch 的相同通道进行归一化。

Batch Norm 1D/2D

训练期公式:

$$\hat{x} = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} \cdot \gamma + \beta \tag{7.1}$$

其中 $\mathrm{E}[x]$ 和 $\mathrm{Var}[x]$ 是在当前 Batch 和空间维度（若是 2D）上计算的均值和方差。**推理期 (Eval)**: 使用训练期间积累的运行均值 (Running Mean) 和运行方差 (Running Var)。

```
nn.BatchNorm2d(num_features)
```

7.2 Layer Normalization (层归一化)

针对单个样本的所有通道进行归一化，常用于 Transformer 等变长序列模型。

Layer Norm

$$y = \frac{x - \mathrm{E}_{layer}[x]}{\sqrt{\mathrm{Var}_{layer}[x] + \epsilon}} \cdot \gamma + \beta \qquad (7.2)$$

均值和方差在 `normalized_shape` 指定的维度上计算。

```
nn.LayerNorm(normalized_shape)
```

7.3 归一化维度对比

- **BatchNorm**: 跨样本计算 (Across N dimension)。
- **LayerNorm**: 跨通道/特征计算 (Across C, H, W dimensions for a single sample)。
- **InstanceNorm**: 针对单个通道、单个样本计算。

Part III

目标篇 (Objectives)

Chapter 8

距离与相似度度量

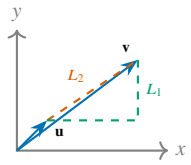


Figure 8.1: Distance Metrics

度量空间是机器学习的基础。在优化模型时，我们需要数学工具来衡量两个向量或分布的相似程度。

8.1 欧氏距离与均方误差 (MSE)

在欧几里得空间中，两点之间的直线距离。

MSE / L2 Loss

$$\text{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \frac{1}{N}\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2 \tag{8.1}$$

从概率学角度，MSE 对应于目标变量服从高斯分布时的最大似然估计。

```
nn.MSELoss()
```

8.2 曼哈顿距离与 L1 Loss

计算绝对误差，对异常值（Outliers）比 MSE 更具鲁棒性。

L1 Loss / MAE

$$\text{MAE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| = \frac{1}{N}\|\mathbf{y} - \hat{\mathbf{y}}\|_1 \tag{8.2}$$

对应于目标变量服从拉普拉斯分布时的最大似然估计。

```
nn.L1Loss()
```

8.3 余弦相似度 (Cosine Similarity)

用于衡量两个向量方向的差异，不考虑向量的绝对大小。常用于文本嵌入（Embeddings）匹配。

Cosine Similarity

$$\text{CosSim}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|_2 \|\mathbf{v}\|_2} = \frac{\sum u_i v_i}{\sqrt{\sum u_i^2} \sqrt{\sum v_i^2}} \tag{8.3}$$

余弦距离定义为 1 – CosSim。

```
nn.CosineSimilarity(dim)
```

Chapter 9

信息论与概率损失

损失函数用于衡量模型预测分布与真实分布之间的差异。在分类生成任务中，基于信息论（Information Theory）的度量是最基础的法则。

9.1 香农熵与交叉熵 (Entropy & Cross-Entropy)

香农熵（Entropy）衡量了一个分布的不确定性。当我们使用模型预测的分布 Q 来逼近真实分布 P 时，需要的额外信息量就是交叉熵。

$$H(P, Q) = -\sum_{x} P(x) \log Q(x) \tag{9.1}$$

$$= H(P) + D_{KL}(P \| Q) \tag{9.2}$$

其中 $H(P)$ 是真实分布的固有熵。在独热编码（One-Hot）下，真实类别概率 $P(x_i) = 1$，固有熵为 0，交叉熵退化为：

$$\mathcal{L}_{CE} = -\log(\hat{y}_c) \tag{9.3}$$

这正是 `nn.NLLLoss(LogSoftmax(x))` 的底层逻辑。

`nn.CrossEntropyLoss()`

9.2 KL 散度 (Kullback-Leibler Divergence)

用于衡量两个概率分布（通常为预测与目标或先验）之间的“距离”（相对熵，不对称）。

KL

$$D_{KL}(P\|Q) = \sum_{x} P(x) \log\left(\frac{P(x)}{Q(x)}\right) \quad (9.4)$$

在 VAE 中，KL 散度常被用于正则化潜变量（Latent Variable）分布使其逼近标准正态分布 $\mathcal{N}(0, I)$。

```
nn.KLDivLoss(reduction='batchmean')
```

9.3 二元交叉熵 (BCE)

多标签分类或二分类的专用交叉熵。

BCE Loss

$$\mathcal{L}_{BCE} = -\frac{1}{N}\sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \quad (9.5)$$

这是伯努利分布下最大似然估计的直接结果。

```
nn.BCELoss() / BCEWithLogitsLoss()
```

Part IV

动力篇 (Dynamics)

Chapter 10

自动求导与计算图

PyTorch 的核心引擎是 Autograd，它实现了自动化的反向传播。

10.1 计算图与链式法则

深度学习模型可以看作是有向无环图（DAG），其中节点是张量，边是数学运算。

10.2 Vector-Jacobian Product (VJP)

在反向传播中，我们需要计算输出对输入的导数。对于函数 $\mathbf{y} = f(\mathbf{x})$，其雅可比矩阵 (Jacobian) J 定义为 $J_{ij} = \frac{\partial y_i}{\partial x_j}$。

然而，PyTorch 并不显式计算整个矩阵（因为对于大型参数，这会导致内存溢出），而是计算 **Vector-Jacobian Product**。

VJP

设上一层传回的梯度向量为 $\mathbf{v} = \frac{\partial L}{\partial \mathbf{y}}$，则当前层的梯度计算为：

$$\mathbf{v}^T J = \left[\frac{\partial L}{\partial y_1}, \ldots, \frac{\partial L}{\partial y_m}\right] \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \tag{10.1}$$

结果即为 $\frac{\partial L}{\partial \mathbf{x}}$。

10.3 PyTorch 操作

- `requires_grad=True`: 标记该张量需要追踪操作以计算梯度。

- `backward()`: 触发从当前节点开始的反向传播。
- `grad`: 存储计算得到的梯度值。

Chapter 11

优化算法

优化算法决定了模型如何根据梯度更新权重。

11.1 随机梯度下降 (SGD)

SGD

更新公式:

$$v_t = \mu v_{t-1} + g_t \tag{11.1}$$

$$\theta_t = \theta_{t-1} - \eta v_t \tag{11.2}$$

其中 μ 是动量因子，g_t 是当前梯度，η 是学习率。

```
torch.optim.SGD(params,
    lr, momentum)
```

11.2 Adam (Adaptive Moment Estimation)

结合了动量（Momentum）和自适应学习率（RMSProp）。

Adam

计算步骤:

1. 一阶矩更新:
   $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$
2. 二阶矩更新:
   $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
3. 偏差修正: $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$, $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$
4. 参数更新: $\theta_t = \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$

```
torch.optim.Adam(params,
lr, betas=(0.9, 0.999))
```

11.3 AdamW (Weight Decay Fix)

在 Adam 中，直接应用 L2 正则化会导致权重衰减效果不如 SGD。AdamW 将权重衰减直接作用于更新步骤。

AdamW

核心差异:

$$\theta_t = \theta_{t-1} - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon} + \lambda \theta_{t-1} \right) \tag{11.3}$$

其中 λ 是权重衰减系数。

```
torch.optim.AdamW(params,
    lr, weight_decay)
```

Chapter 12

随机过程 (Stochastic Processes)

随机过程是处理时间序列、扩散模型前向与反向加噪的核心数学工具。我们将在此探讨布朗运动与随机微分方程（SDE）。

12.1 布朗运动与维纳过程 (Brownian Motion & Wiener Process)

维纳过程（Wiener Process）W_t 是最基础的连续时间随机过程。其增量 $W_{t+\Delta t} - W_t$ 服从正态分布 $\mathcal{N}(0, \Delta t)$。

$$dW_t \sim \mathcal{N}(0, dt) \tag{12.1}$$

```
torch.randn(size) * torch.sqrt(dt)
```

12.2 随机微分方程 (Stochastic Differential Equations)

在扩散模型（如 Score-based Generative Models）中，数据受到由 SDE 控制的演化：

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)dt + g(t)d\mathbf{w} \tag{12.2}$$

其中 $\mathbf{f}(\mathbf{x}, t)$ 是漂移系数（Drift），$g(t)$ 是扩散系数（Diffusion）。

- **(Euler-Maruyama Method)**

对于 SDE $dx_t = f(x_t, t)dt + g(t)dw_t$，其数值离散解为：

$$x_{t+\Delta t} = x_t + f(x_t, t)\Delta t + g(t)\sqrt{\Delta t}\cdot\epsilon, \quad \epsilon \sim \mathcal{N}(0, 1) \tag{12.3}$$

PyTorch 实现:

```
# x: current state, t: time, dt: step
eps = torch.randn_like(x)
drift = f(x, t) * dt
diffusion = g(t) * torch.sqrt(dt) * eps
x_next = x + drift + diffusion
```

12.3 SDE 轨迹 (SDE Trajectories)

随机微分方程的每一次采样都会产生一条不同的轨迹（Trajectory）。

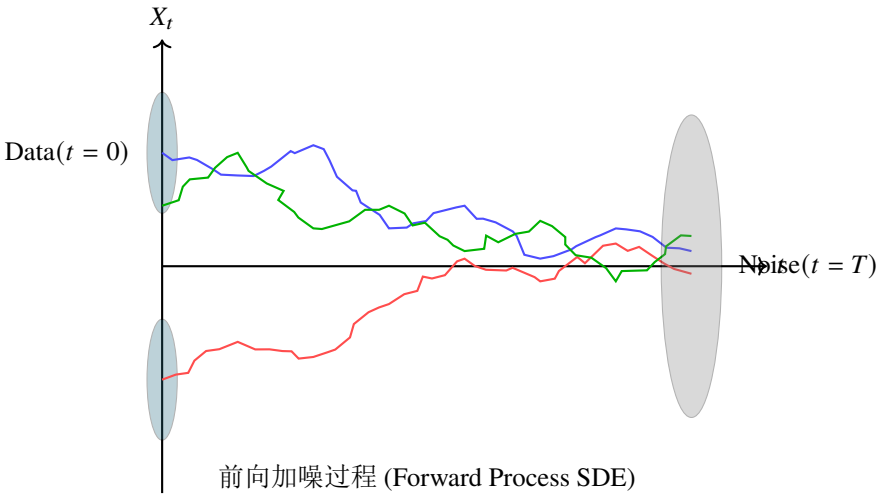


Figure 12.1: 数据分布通过随机微分方程演化为噪声分布的可能轨迹

轨迹的演化展示了扩散模型前向加噪过程的几何直觉，数据逐步从复杂分布演化向各向同性的高斯分布。

Part V

架构篇 (Architectures)

Chapter 13

注意力机制

注意力机制是现代深度学习（尤其是 Transformer 模型）的灵魂，它允许模型动态地关注输入序列的不同部分。

13.1 缩放点积注意力 (Scaled Dot-Product Attention)

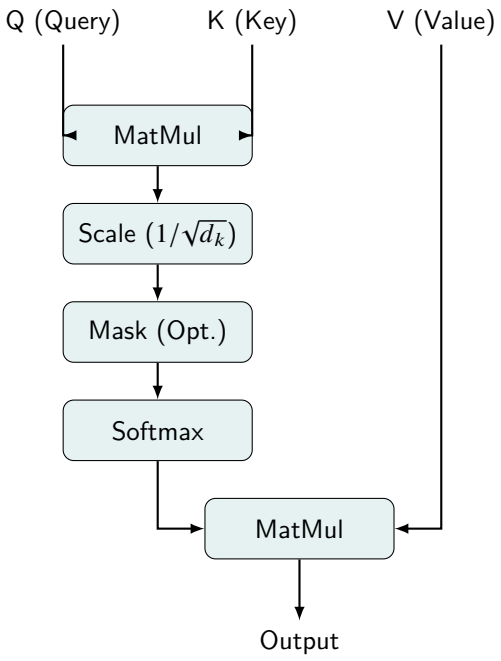


Figure 13.1: 缩放点积注意力的数据流图 (Scaled Dot-Product Attention)

Self-Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{13.1}$$

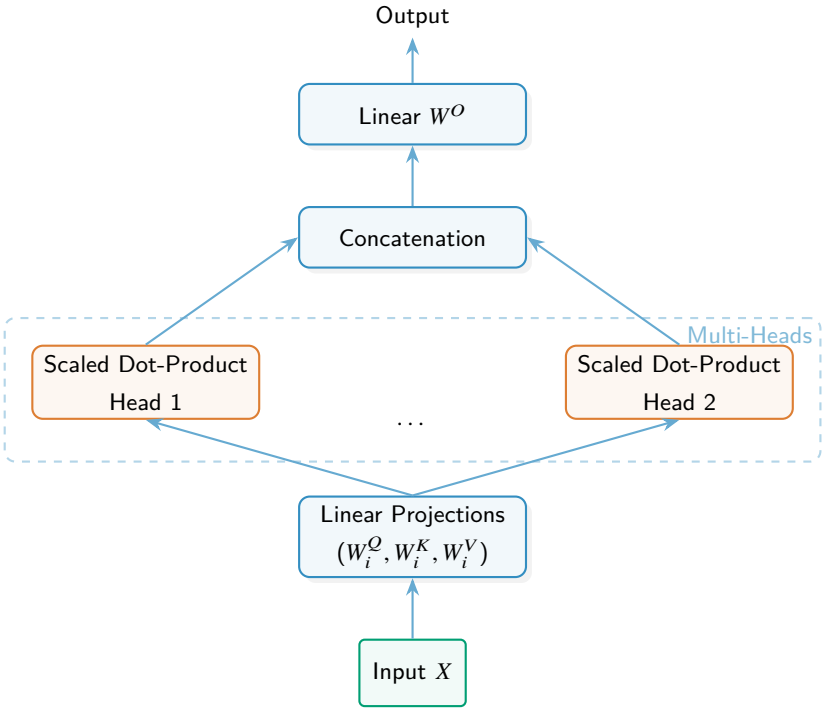
其中：

- Q (Query), K (Key), V (Value) 是线性变换后的矩阵。
- d_k 是 Key 的维度，除以 $\sqrt{d_k}$ 是为了防止点积结果过大导致梯度消失。

```
torch.nn.functional.scaled_dot_product_attention
```

13.2 多头注意力 (Multi-Head Attention)

多头注意力通过并行运行多个自注意力头，使模型能够同时从不同的表示子空间捕捉信息。



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{13.2}$$

其中 $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

```
nn.MultiheadAttention(embed_dim, num_heads)
```

Chapter 14

生成模型数学基础

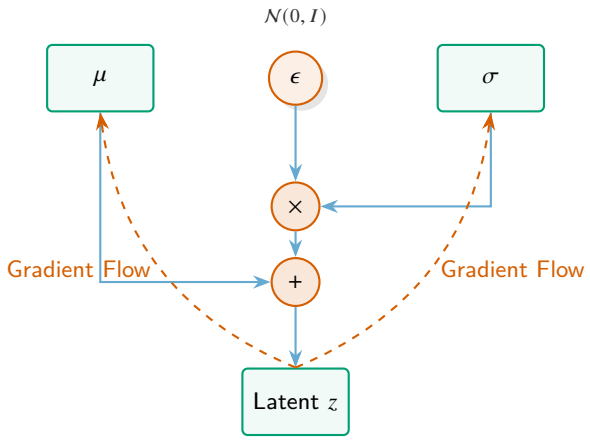
生成模型旨在学习数据的概率分布 $p(x)$，从而生成新的样本。

14.1 VAE (Variational Autoencoder)

VAE 通过最大化变分下界（ELBO）来训练。

14.1.1 重参数化技巧 (Reparameterization Trick)

为了让随机采样过程可导，我们将随机性转移到外部噪声。这使得梯度能够流过分布参数 μ 和 σ。



$$\mathbf{z} = \mu + \sigma \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I) \tag{14.1}$$

```
z = mu + sigma * torch.randn_like(sigma)
```

14.1.2 ELBO 损失函数

VAE 的目标函数由两部分组成：重建误差和正则化项。

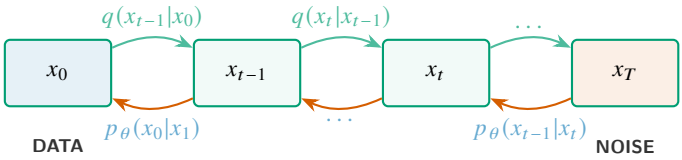
VAE

$$\mathcal{L}_{VAE} = \mathbb{E}_{q(z|x)}[\log p(x|z)] - D_{KL}(q(z|x)\|p(z)) \tag{14.2}$$

其中第一项是重建似然（通常用 MSE 或 BCE），第二项是隐藏变量分布与先验分布（通常为标准正态）的 KL 散度。

14.2 Diffusion Model (扩散模型) 简介

扩散模型通过逐渐向数据添加噪声（正向过程）和学习去噪（逆向过程）来生成图像。



DDPM

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \tag{14.3}$$

通过重参数化，我们可以直接从 x_0 计算 x_t。

Part VI

大模型纪元 (Foundation Models)

Chapter 15

大模型组件 (LLM Components)

现代大语言模型（如 LLaMA、Mixtral）在 Transformer 基础架构上引入了更高效的算子，极大增强了模型的外推能力与推理速度。

15.1 RMSNorm (Root Mean Square Normalization)

抛弃了均值平移（Mean-centering），仅使用均方根缩放，从而节省了计算开销而性能基本无损。

RMSNorm

$$\text{RMSNorm}(x) = \frac{x}{\text{RMS}(x)} \odot \gamma \quad \text{其中} \text{RMS}(x) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \epsilon} \tag{15.1}$$

这里 γ 是可学习的缩放参数，ϵ 为防止除零的微小常数。

```
x / torch.sqrt(torch.mean(x**2) + eps) * weight
```

15.2 SwiGLU (Swish Gated Linear Unit)

LLaMA 中的默认激活机制，采用 Swish 变体控制信息流。

SwiGLU

$$\text{SwiGLU}(x, W, V) = \text{Swish}(xW) \odot xV \tag{15.2}$$

其中 Swish 函数定义为:

$$\text{Swish}_\beta(z) = z \cdot \sigma(\beta z) = \frac{z}{1 + e^{-\beta z}} \tag{15.3}$$

当 $\beta = 1$ 时，即为 SiLU 函数。

15.3 RoPE (Rotary Position Embedding)

旋转位置编码通过将绝对位置注入为旋转矩阵，自然推导出了相对位置衰减特性。

RoPE ( )

对于位置 m 的特征向量 $\mathbf{x} = [x_1, x_2, \ldots, x_d]$，将其两两分组应用旋转矩阵：

$$f(\mathbf{x}, m) = \mathbf{x} R^d_{\Theta,m} \tag{15.4}$$

旋转矩阵 $R^d_{\Theta,m}$:

$$\begin{bmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \ldots \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \ldots \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \ldots \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \ldots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{15.5}$$

其中基频 $\theta_i = 10000^{-2(i-1)/d}$。

15.4 MoE (Mixture of Experts)

通过路由机制（Routing）让每个 Token 仅激活部分参数（如 Top-2），实现参数量扩大而不增加前向推理时间。

Top-K

路由权重: 对于输入 x，专家 E_i 的权重为：

$$G(x)_i = \begin{cases} \frac{e^{x \cdot W_{r,i}}}{\sum_{j \in T} e^{x \cdot W_{r,j}}}, & \text{if } i \in T \\ 0, & \text{otherwise} \end{cases} \tag{15.6}$$

其中 T 是根据 $x \cdot W_r$ 选出的 Top-K 索引集合。

最终输出:

$$y = \sum_{i=1}^{N} G(x)_i E_i(x) \tag{15.7}$$

Chapter 16

状态空间模型 (SSM & Mamba)

状态空间模型（State Space Models，如 S4、Mamba）是试图替代 Transformer 作为序列建模基石的架构，它结合了 RNN 的推理效率和 CNN 的训练并行性。

16.1 连续系统数学模型 (ODE)

状态空间模型本质上是将输入信号 $x(t) \in \mathbb{R}$ 映射到隐藏状态 $h(t) \in \mathbb{R}^N$，再投影到输出 $y(t) \in \mathbb{R}$ 的常微分方程。

状态方程:

$$h'(t) = \mathbf{A}h(t) + \mathbf{B}x(t) \tag{16.1}$$

输出方程:

$$y(t) = \mathbf{C}h(t) + \mathbf{D}x(t) \tag{16.2}$$

其中 $\mathbf{A} \in \mathbb{R}^{N \times N}$ 是演化矩阵，通常使用 HiPPO（High-order Polynomial Projection Operators）初始化，以确保系统能够“记住”历史。

16.2 离散化 (Discretization: ZOH)

在实际的深度学习处理中，信号是离散的文本 Token（时间间隔 Δ）。我们需要使用零阶保持（Zero-Order Hold, ZOH）将其离散化为类似 RNN 的差分方程形式。

ZOH

给定步长 Δ，离散系统参数 $\bar{\mathbf{A}}$ 和 $\bar{\mathbf{B}}$ 的严格数学推导为：

$$\bar{\mathbf{A}} = \exp(\Delta\mathbf{A}) \tag{16.3}$$

$$\bar{\mathbf{B}} = (\Delta\mathbf{A})^{-1}(\exp(\Delta\mathbf{A}) - I) \cdot \Delta\mathbf{B} \tag{16.4}$$

离散递推公式:

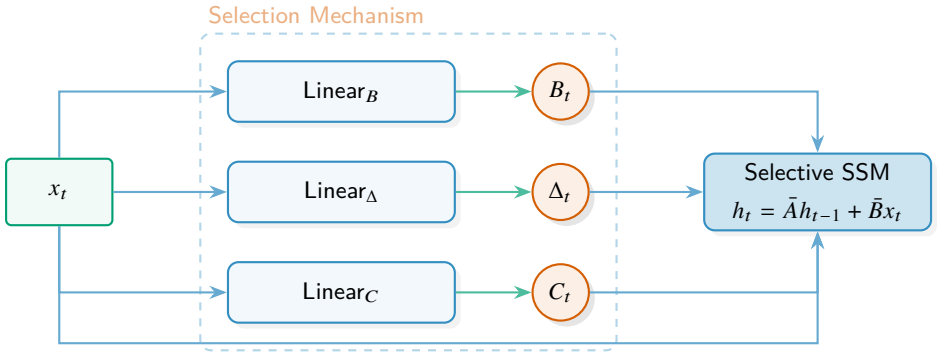
$$h_k = \bar{\mathbf{A}}h_{k-1} + \bar{\mathbf{B}}x_k \tag{16.5}$$

$$y_k = \mathbf{C}h_k \tag{16.6}$$

(省略 D 项，通常在网络中视为残差连接。)

16.3 选择性扫描 (Mamba: Selective Scan)

在传统的 S4 中，$\bar{\mathbf{A}}, \bar{\mathbf{B}}$ 对于所有时间步是 ** 时不变的（LTI）**，因此可以通过 FFT（快速傅里叶变换）进行卷积加速。而 Mamba 的核心突破在于 ** 数据相关性（Data-Dependent）**：让 $\Delta, \mathbf{B}, \mathbf{C}$ 成为输入 x_t 的函数，从而能够选择性地过滤或记住信息（类似于 LSTM 的遗忘门）。



Mamba

由于时变特性，FFT 不再适用。Mamba 使用高效的前缀和扫描算法（Hardware-aware Scan）：

$$\mathbf{B}_t = \text{Linear}_B(x_t), \quad \mathbf{C}_t = \text{Linear}_C(x_t) \tag{16.7}$$

$$\Delta_t = \text{Softplus}(\text{Linear}_\Delta(x_t)) \tag{16.8}$$

此时 h_t 的递推变为时变的：

$$h_t = \bar{\mathbf{A}}_t h_{t-1} + \bar{\mathbf{B}}_t x_t \tag{16.9}$$

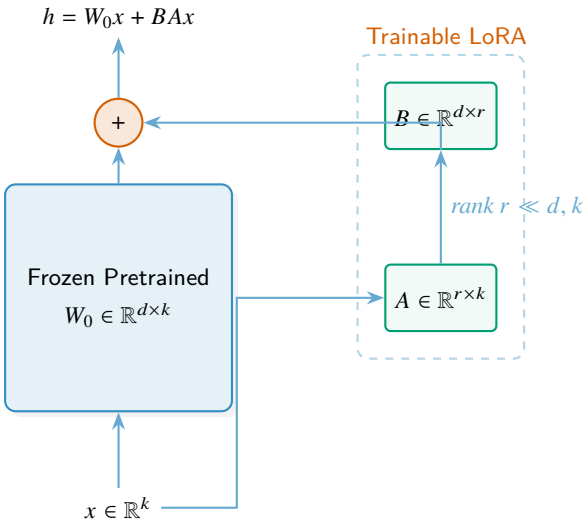
Chapter 17

参数高效微调 (PEFT & LoRA)

当模型规模增长至百亿甚至千亿参数时，全参微调（Full Fine-Tuning）的显存和计算成本难以承受。低秩自适应（Low-Rank Adaptation, LoRA）通过极少的额外参数，保留了原始知识并适应新任务。

17.1 矩阵低秩分解

LoRA 的核心思想是冻结预训练模型的权重矩阵，并通过旁路的两个低秩矩阵之积来表示权重的增量（ΔW）。



LoRA

设预训练权重矩阵为 $W_0 \in \mathbb{R}^{d\times k}$，冻结不变。更新量 ΔW 通过两个低秩矩阵 $B \in \mathbb{R}^{d\times r}$ 和 $A \in \mathbb{R}^{r\times k}$ 乘积得到：

$$W = W_0 + \Delta W = W_0 + BA \tag{17.1}$$

其中秩 $r \ll \min(d, k)$，例如 $r = 8$ 或 $r = 64$。

17.2 前向传播与初始化

17.2.1 前向计算逻辑

对于输入 $\mathbf{x} \in \mathbb{R}^k$，全连接层的原本输出为 $h = W_0\mathbf{x}$。引入 LoRA 之后，前向传播变为两个并行的矩阵乘法最后相加。

LoRA

$$h = W_0\mathbf{x} + \frac{\alpha}{r}BA\mathbf{x} \tag{17.2}$$

其中 α 是一个缩放超参数，在初始阶段控制新加入参数对最终结果的影响。当秩 r 改变时，可以通过 $\frac{\alpha}{r}$ 保持方差稳定。

```
y = F.linear(x, W0) +
F.linear(x, B @ A) * (alpha / r)
```

17.2.2 零初始化技巧

为了保证在训练刚开始时（即第一个 Step 前），引入的 LoRA 模块不会破坏预训练模型的行为，矩阵 A 和 B 的初始化至关重要。

LoRA

矩阵 A: 使用标准正态分布初始化（Gaussian / Kaiming）。

$$A \sim \mathcal{N}(0, \sigma^2) \tag{17.3}$$

矩阵 B: 必须使用全零矩阵初始化。

$$B = \mathbf{0} \implies BA = \mathbf{0} \tag{17.4}$$

这样，初始状态下 $\Delta W = 0$，保证了 $h = W_0\mathbf{x}$ 与原始模型输出完全一致。

17.3 推理合并 (Merging Weights)

在推理阶段，由于矩阵乘法的分配律，我们可以直接将 BA 加回到 W_0 中。因此，部署带有 LoRA 的大模型，其 ** 推理延迟与基座模型完全相同 **（Zero Latency Cost）。

LoRA

$$W_{deploy} = W_0 + BA \tag{17.5}$$

在 PyTorch 中只需执行 `W0.add_(B @ A)` 即可卸载独立参数。

Part VII

前沿探索 (The Frontiers)

Chapter 18

对抗与对比学习

传统的预测任务旨在最小化误差，而对抗与对比学习则是让模型在数据分布与表征空间中互相博弈或聚类。

18.1 生成对抗网络 (GAN)

GAN 由生成器 (Generator) 和判别器 (Discriminator) 组成，两者进行极小极大博弈 (Min-Max Game)。

GAN

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1-D(G(z)))] \tag{18.1}$$

判别器 D 试图将原数据打高分 (1)，假数据打低分 (0)；生成器 G 试图骗过 D（让 $D(G(z))$ 趋近于 1）。

18.2 对比学习与 InfoNCE 损失

多模态模型（如 CLIP）和自监督学习的核心，通过拉近正样本、推开负样本来学习高维特征。

InfoNCE (Temperature Scaled Cross-Entropy)

$$\mathcal{L}_q = -\log \frac{\exp(\mathbf{q} \cdot \mathbf{k}^+/\tau)}{\sum_{i=0}^{K} \exp(\mathbf{q} \cdot \mathbf{k}_i/\tau)} \tag{18.2}$$

其中 $\mathbf{q}$ 是查询向量，$\mathbf{k}^+$ 是匹配的正样本，τ 是温度超参数（控制对困难负样本的惩罚力度）。

```
F.cross_entropy(logits / temperature, labels)
```

Chapter 19

图网络与空间视觉

超越传统的网格数据（像素），神经网络能够处理非欧几里得空间（图）或以全新视角审视视觉（Patch）。

19.1 图卷积网络 (GCN)

在图数据上进行信息传递（Message Passing），节点特征通过邻接矩阵进行聚合。

GCN

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \quad (19.1)$$

其中：

- $\tilde{A} = A + I_N$ 是带有自环的邻接矩阵（即每个节点也看自己）。
- $\tilde{D}$ 是 $\tilde{A}$ 的度矩阵（对角阵，用于归一化，防止度大的节点梯度爆炸）。
- $H^{(l)}$ 是第 l 层的节点特征矩阵。

19.2 视觉 Transformer (ViT)

将图像视为序列 (Sequence of Patches)，彻底打破了 CNN 的归纳偏置。

ViT Patch

一张 $H \times W \times C$ 的图像被切割为 N 个大小为 $P \times P$ 的图像块。

$$N = \frac{H \times W}{P^2} \tag{19.2}$$

数学展开:

$$\mathbf{z}_0 = [x_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \ldots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{pos} \tag{19.3}$$

其中 $\mathbf{x}_p^i$ 是展平的图像块，$\mathbf{E}$ 是线性投影矩阵，$\mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D}$ 是可学习的 1D 位置编码，x_{class} 是用于分类的全局 Token。

Chapter 20

对齐与极端量化

当大模型具备涌现能力后，下一步是让其输出符合人类偏好（对齐），并能在消费级硬件上运行（量化）。

20.1 直接偏好优化 (DPO)

彻底绕过了传统 RLHF 中复杂的强化学习 (PPO) 训练流程，直接在偏好数据上优化语言模型本身。

DPO

设 π_θ 为正在训练的模型，π_{ref} 为参考模型，y_w 为人类偏好的回答，y_l 为被拒绝的回答。

$$\mathcal{L}_{DPO} = -\mathbb{E}_{(x,y_w,y_l)}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w|x)}{\pi_{ref}(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_{ref}(y_l|x)}\right)\right] \tag{20.1}$$

其中 β 是控制偏离参考模型程度的温度系数，σ 为 Sigmoid 函数。该公式隐式定义了奖励。

20.2 QLoRA 与 NF4 量化

为了在单张显卡上微调大模型，QLoRA 引入了一种信息论上最优的新数据类型：4-bit NormalFloat (NF4)。

NF4 的数学基础: 假设神经网络的权重服从均值为 0 的正态分布 $\mathcal{N}(0, \sigma^2)$。NF4 预先计算了该分布的 16 个等概率分位数作为量化点.

$$q_i = \frac{1}{2}\left(Q\left(\frac{i}{16}\right) + Q\left(\frac{i+1}{16}\right)\right) \tag{20.2}$$

双重量化 (Double Quantization): 为了存储量化所需的缩放常数 c，对这些常数进行二次 8-bit 量化，使得平均每参数的内存开销再降低 0.37 bits。

Chapter 21

下一代生成法则

超越经典的 VAE 与基础扩散模型，当代生成大模型（图像、视频）采用了更强大的引导机制与流模型数学框架。

21.1 无分类器引导 (CFG - Classifier-Free Guidance)

Stable Diffusion 系列模型“听懂”人类 Prompt 的绝对核心数学技巧。

CFG

在每个去噪步 t，模型同时预测有条件（Prompt c）和无条件（空文本 $\emptyset$）的噪声。

$$\hat{\epsilon}_\theta(x_t, c) = \epsilon_\theta(x_t, \emptyset) + w \cdot (\epsilon_\theta(x_t, c) - \epsilon_\theta(x_t, \emptyset)) \tag{21.1}$$

其中 w 即为 `guidance_scale`。当 $w > 1$ 时，它在几何空间中强行放大了向条件 c 靠近的方向向量，从而生成高度匹配 Prompt 但可能略微降低多样性的图像。

21.2 流匹配与整流 (Flow Matching & Rectified Flows)

Sora 和 Stable Diffusion 3 抛弃了传统的 DDPM 马尔可夫链，转向基于连续常微分方程 (ODE) 的直接轨迹预测。

Rectified Flows (ODE)

建立一条从纯噪声 $X_0 \sim \mathcal{N}(0, I)$ 到真实数据 X_1 的直线轨迹：

$$X_t = tX_1 + (1-t)X_0, \quad t \in [0,1] \tag{21.2}$$

神经网络 $v_\theta(X_t, t)$ 的目标是直接预测这条直线的导数（即流场）：

$$\mathcal{L} = \mathbb{E}_{t,X_0,X_1}\left[\|v_\theta(X_t,t) - (X_1 - X_0)\|_2^2\right] \tag{21.3}$$

这比扩散模型试图预测加入的随机噪声在数学上更加优美，且在采样时能以极少的步数求得 ODE 积分。